Ollscoil na Gaillimhe
University of Galway

CT336/CT404 Graphics & Image Processing

4th year B.Sc. (CS&I.T.).
2nd year M.Sc. (Software Development & Design)
1st year M.Sc. (Biomedical Engineering)
Visiting students

University *of* Galway.ie

Lecture 2: Introduction to 3D Graphics

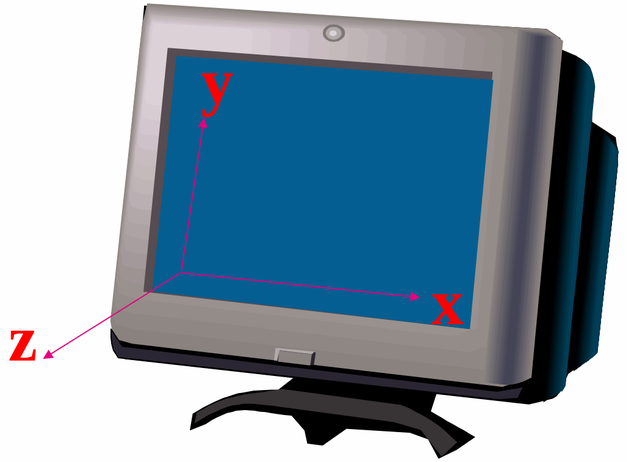
Ollscoil na Gaillimhe
University of Galway

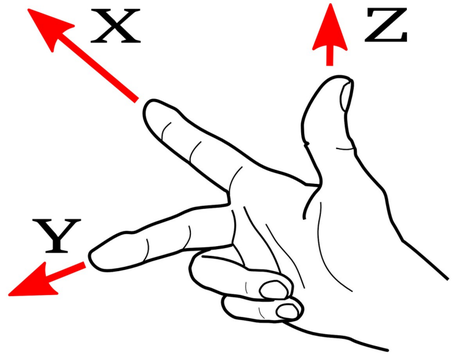
- Last Lecture
  - Graphics Libraries (OpenGL, WebGL, etc.)
  - 2D Transformations (with examples in Canvas 2D)

- Today
  - 3D Coordinate Systems
  - 3D Projections & Transformations
  - Introduction to Three.js
  - Three.js Examples: Primitives and Geometry, Nested Coordinates, Transformations
  - Shading

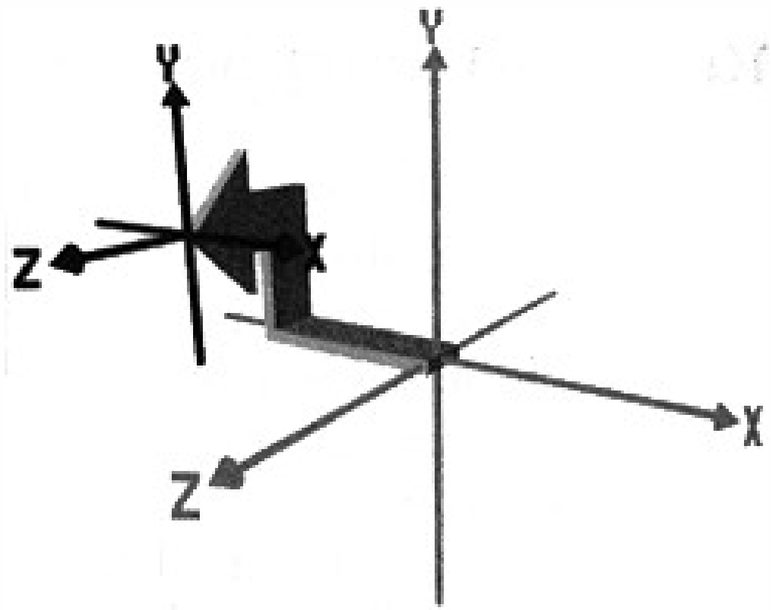
3D Coordinate Systems

- In a 3D coordinate system, a point P is referred to by three real numbers (coordinates): x, y, z
- The directions of x, y, and z are not universally defined, but normally follow the '**right-hand rule**' for axes system
- In this case z defines the coordinate's distance 'out of' the monitor and negative z values go 'into' the monitor
- *Do you remember the coordinate system in Canvas 2D from last lecture?*

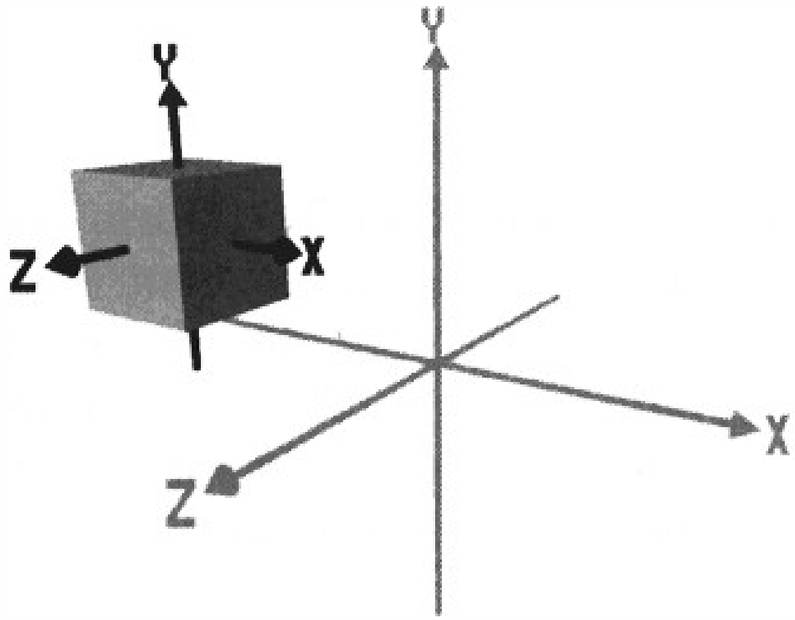




Nested Coordinate Systems



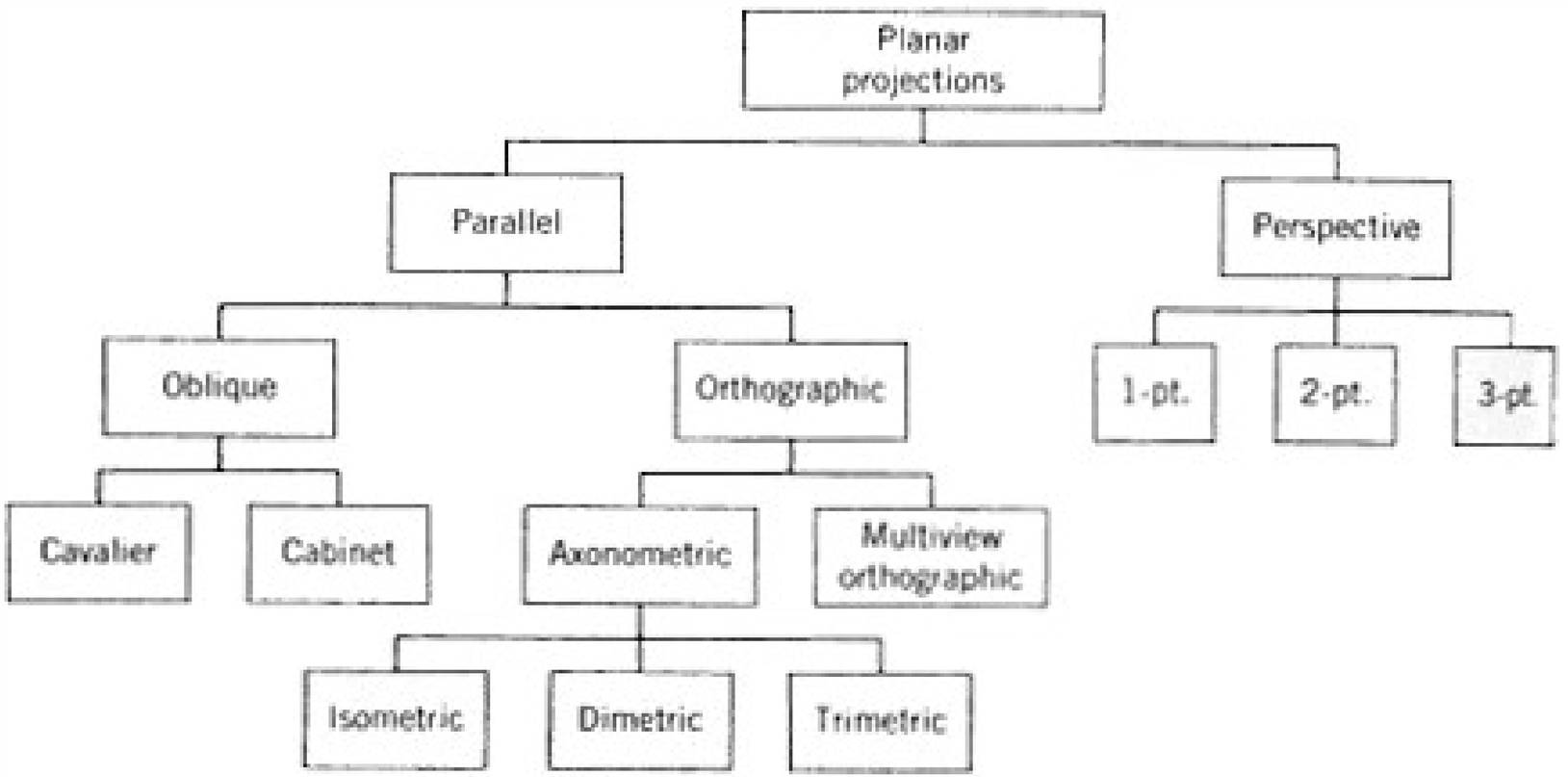
A nested coordinate system defined as a translation relative to the world coordinate system:
For example, -3.0 units along the X axis, 2.0 units along the Y axis, and 2.0 units along the Z axis



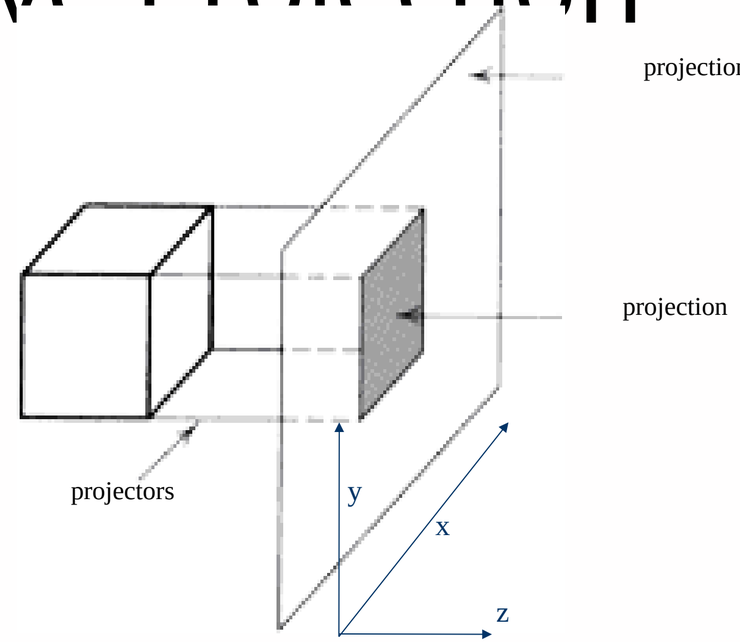
A box shape contained in this nested co-ordinate system centered at its origin (0,0,0)

3D Viewing: Planar Projections

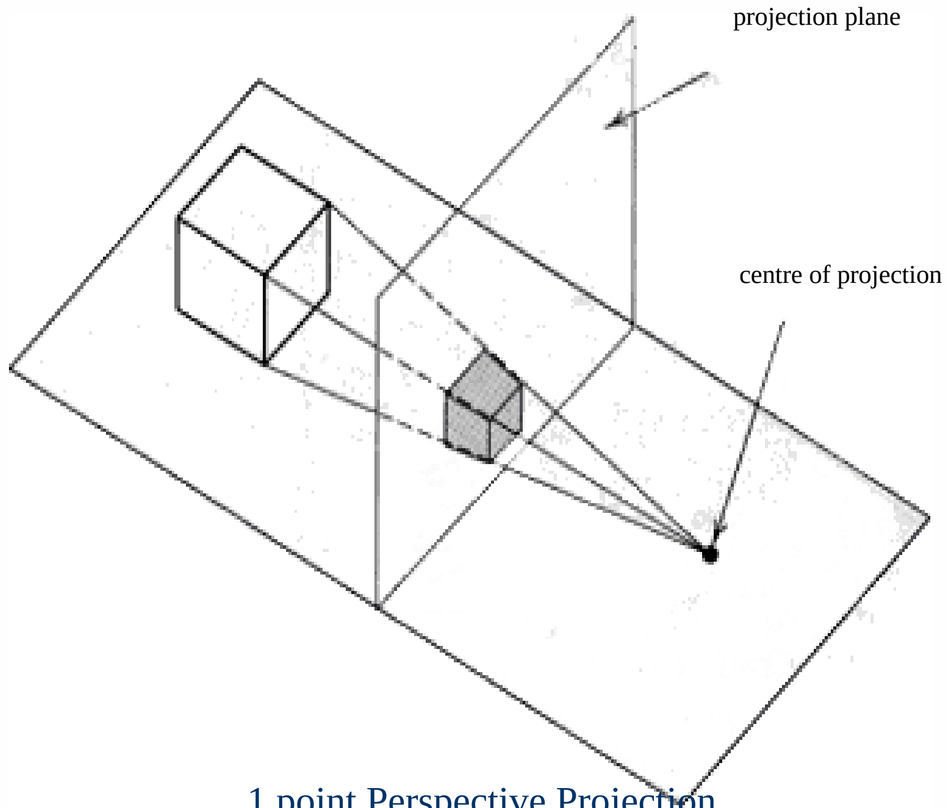
- Why do we need planar projections in 3D graphics?



Parallel Projection and Perspective Projection

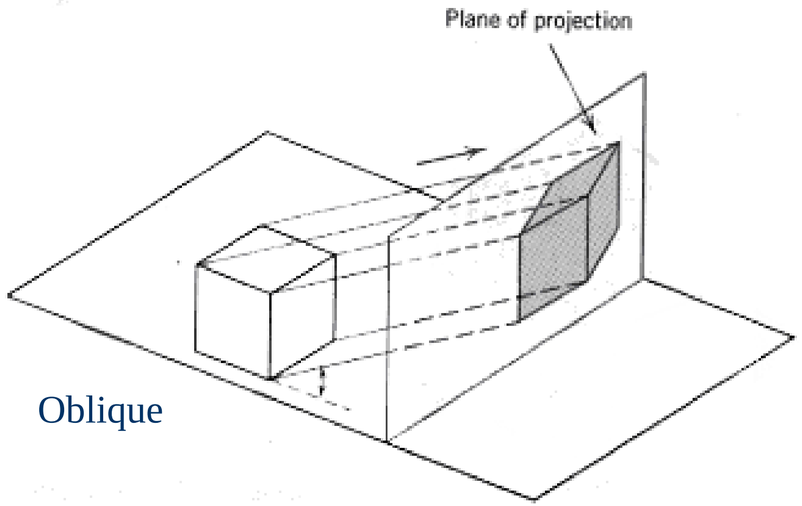


Orthogonal Parallel Projection

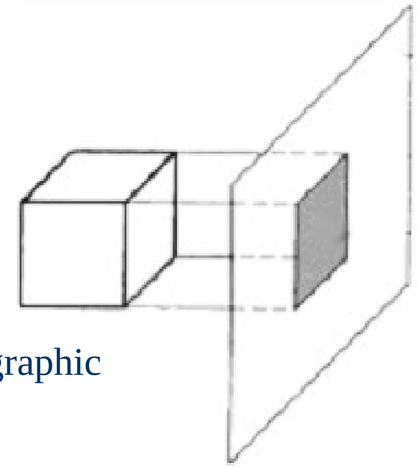


1 point Perspective Projection

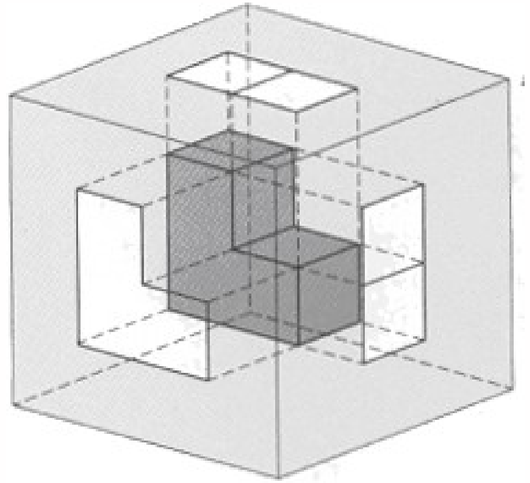
Parallel Projections



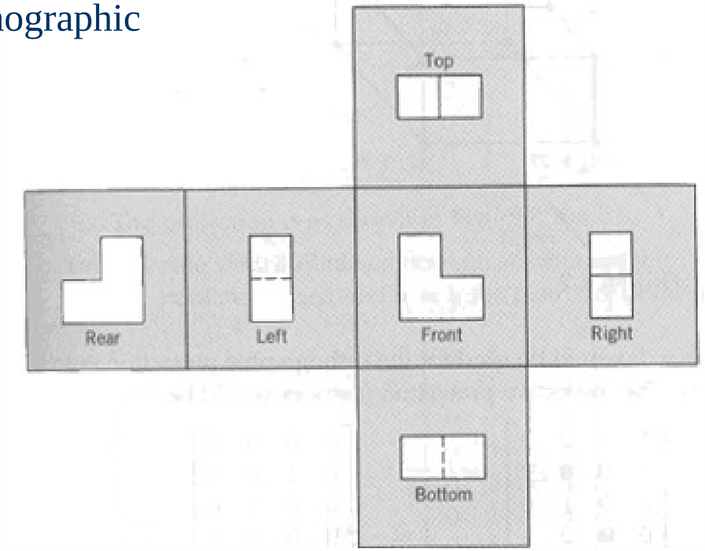
Oblique



Orthographic



Multi-View Orthographic



3D Transformations: Translation

- To translate a 3D point, modify each dimension separately:

  x' = x + a1

  y' = y + a2

  z' = z + a3

$$[x' \;\; y' \;\; z' \;\; 1] = [x \;\; y \;\; z \;\; 1] \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \mathbf{1} \end{bmatrix}$$

3D Rotation about ***principal (body)*** axes

- Principal axis: An imaginary line through the ‘Center of Mass’ of a body around which the body rotates Rotation matrices define rotations by angle α about the principal axes

- Rotation about the x axis is often called ‘pitch’
- Rotation about the y axis is often called ‘yaw’
- Rotation about the z axis is often called ‘roll’

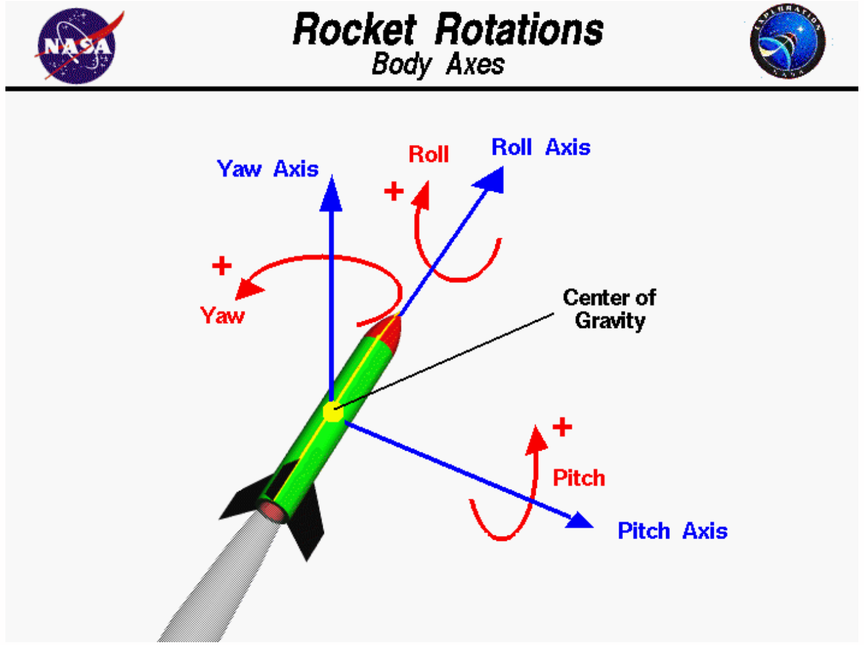


Image source: NASA Glenn Research Centre ‘A beginner’s guide to Aeronautics’

3D Rotation about ***principal (body)*** axes

Ollscoil na Gaillimhe
University of Galway

- Rotation matrices define rotations by angle α about the principal axes

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$

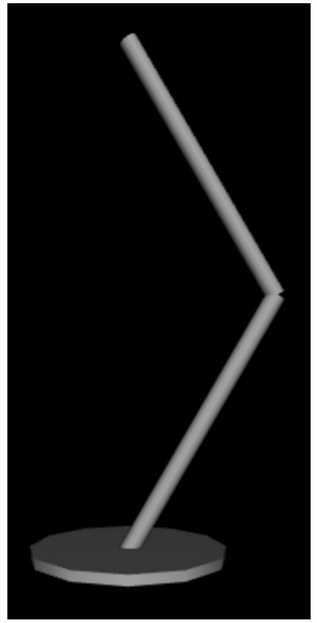
- To get new coordinates after rotation, multiply the point [x y z] by the rotation matrix, e.g.

$$[x'\ y'\ z'] = [x\ y\ z]\ R_X$$

For example, as a point rotates about the x axis, its x component remains unchanged

3D Rotation about ***arbitrary*** axis

- You can rotate about any axis, not just the principal axes.
- You specify a 3D point, and the axis of rotation is defined as the line that joins the origin to this point (e.g. a toy spinning top will rotate about the Y axis, defined as (0, 1, 0))
- You must also specify the amount to rotate by, this is measured as an angle in Radians (i.e. Math.PI*2 Radians is 360 degrees)
- Three.js: the centre of rotation is, by default, the origin of the local coordinate system (To change this, we can create a parent object, position the child relative to that, and perform rotation on the parent (see example below: desk lamp)



Graphics APIs (Application Programming Interfaces)

Ollscoil na Gaillimhe
University of Galway

- Low Level Graphics APIs:
  - Libraries of graphics functions that can be accessed from a standard programming language
  - Procedural rather than descriptive => fast!
  - Procedural: the programmer calls the graphics functions which carry out operations immediately – programmer also has to write all other application code: interface, etc.
  - Examples: OpenGL, DirectX, Vulkan, Java Media APIs
  - Examples that run inside the browser (Javascript): Canvas2D, WebGL, SVG

- High Level Graphics APIs:
  - The programmer describes the required graphics, animations, interactivity etc. and doesn't need to deal with how this will be displayed and updated
  - Descriptive rather than procedural => slower and less flexible, because it is generally interpreted and general purpose rather than task specific
  - Example: VRML/X3D

Graphics API: WebGL and Three.js

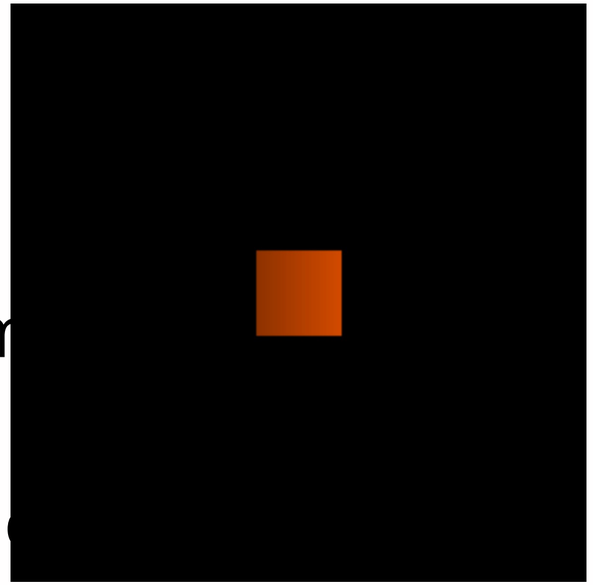
Ollscoil na Gaillimhe
University of Galway

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins. **WebGL** is fully integrated with other web standards, allowing GPU-accelerated usage of physics and image processing and effects as part of the web page canvas.

- Three.js is a cross-browser JavaScript library and Application Programming Interface used to create and display animated 3D computer graphics in a web browser. Three.js uses WebGL.

- Some Resources:
  - https://threejs.org/manual/
  - https://threejs.org/docs/
  - https://codepen.io/rachsmith/post/beginning-with-3d-webgl-pt-1-the-scene

“Hello World” equivalent in Three.js



Threejs-1-hello_cube.html

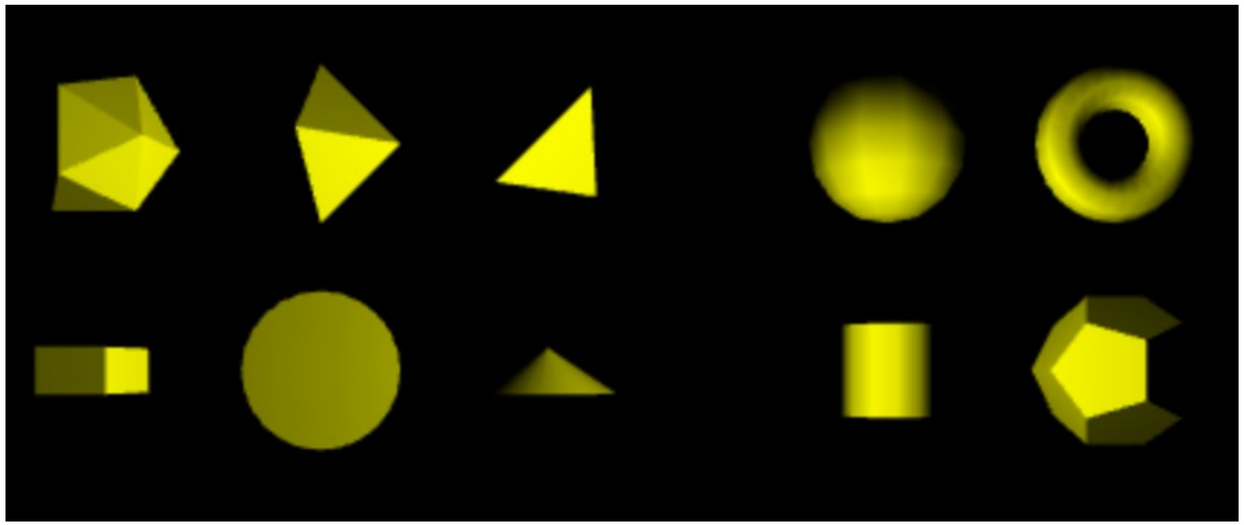
A “hello world” scene with a simple piece of geometry (a cube) in a scene, with a light and a cam

In three.js, a visible object is represented as ‘mesh’ and constructed from a *geometry* and a *material*.

Three.js: 3D primitives

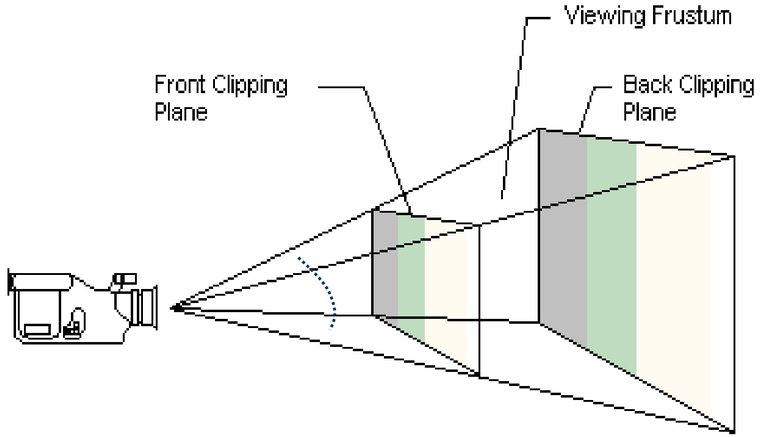
- Three.js provides a range of primitive geometry, as well as the functionality to implement more complex geometry at a lower level
- See: https://threejs.org/manual/?q=prim#en/primitives

- Example: Threejs-2-primitives_galore.html
  - Illustrates many of the “primitives” provided by threejs, as well as other basics such as setting positions

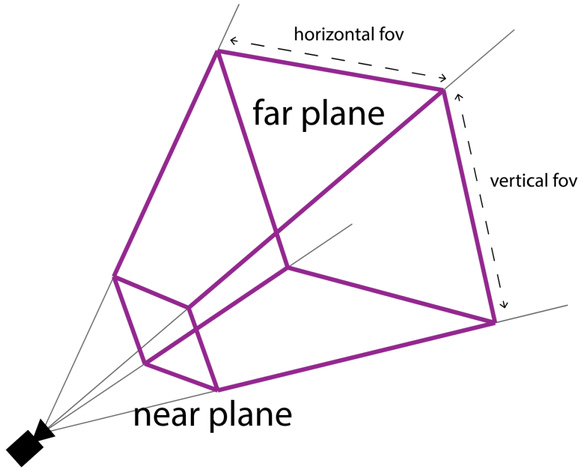


Three.js: Cameras

- 3D Graphics API cameras allow you to define:
  - Camera location (x, y, z)
  - Camera orientation (x rotation, y rotation, z rotation)
  - Viewing Frustum =Field of View (FOV) + clipping planes



- In Three.js the FoV can be differently set in the vertical and horizontal direction via the 1st and 2nd arguments to the constructor (fov, aspect)
- Generally, aspect ratio should match that of the canvas height, width (width/height) or else the scene will be stretched



Three.js/WebGL Lighting

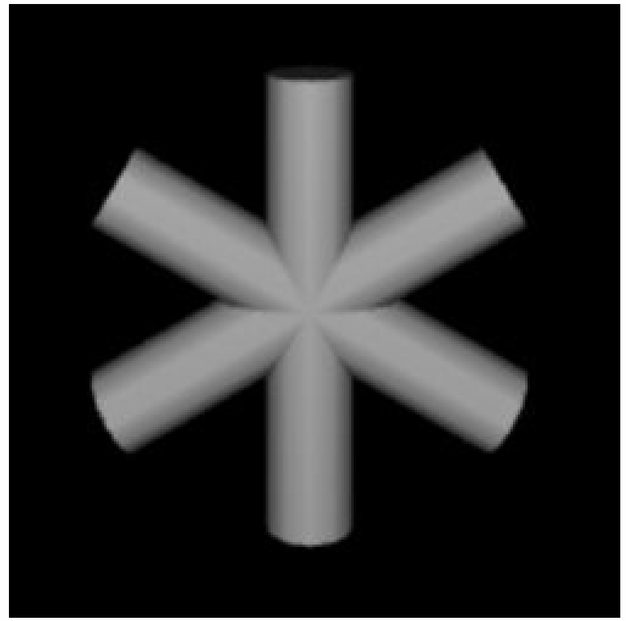
Ollscoil na Gaillimhe
University of Galway

- Six different types of lights are available:

  - **Point lights** - rays emanate in all directions from a 3D point source, (e.g. a lightbulb)
  - **Directional lights** - rays emanate in one direction only from infinitely far away (like the sun)
  - **Spotlights** - project a cone of light from a 3D point source, aimed at a specific target point.
  - **Ambient lights** – simulate in a simplified way the lighting of an entire scene due to complex light/surface interactions; lights up everything regardless of position or occlusion
  - **Hemisphere lights** – ambient lights that affect the ’ceiling’ or ‘floor’ hemisphere of objects rather than affecting them in entirety
  - **RectAreaLights** – emit rectangular areas of light (e.g. fluorescent light strip)

Three.js: Rotation around local origin

- Threejs-4-rotated-cylinders.html



Three.js: Nested Co-ordinates

Ollscoil na Gaillimhe
University of Galway

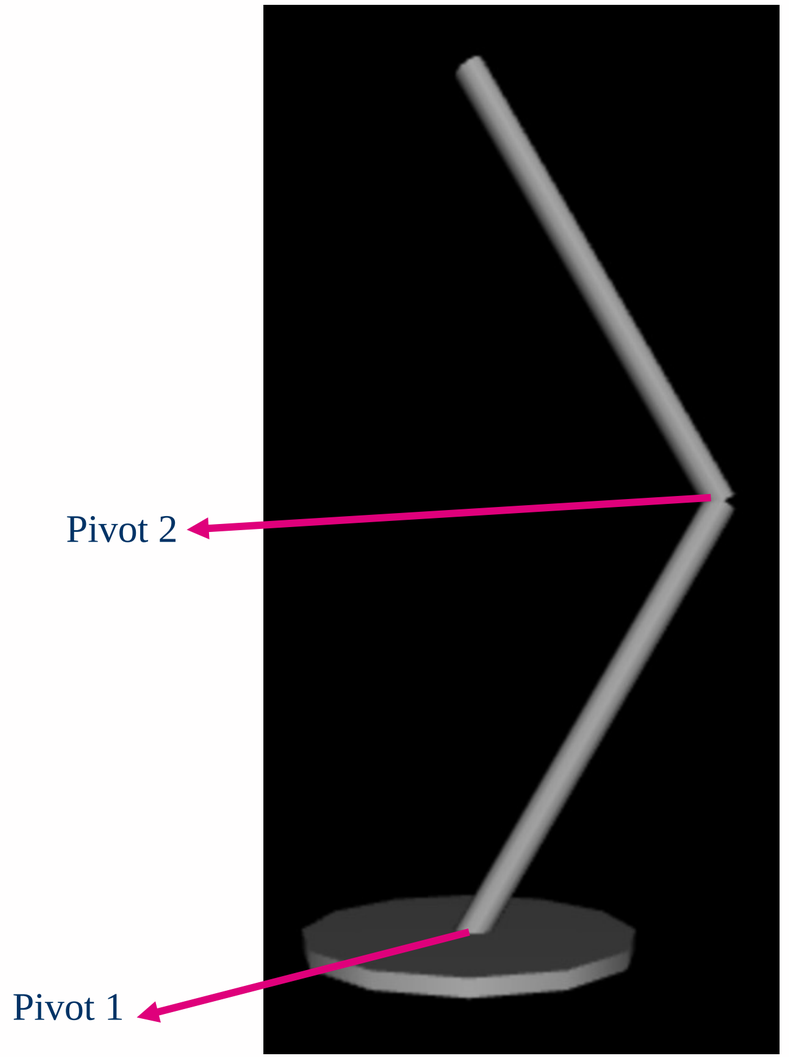
- Consider the steps required to build a room with a table and a lamp on it:
  - Build a lamp, with each component relative to a lamp coordinate system
  - Build a table, with each component relative to a table coordinate system
  - Place the lamp on the table by nesting the lamp coordinate system in the table coordinate system
  - Build a room, with each component relative to a room coordinate system
  - Place the table (and its lamp) by nesting the table coordinate system in the room coordinate system
  - This approach has allowed us to build each piece of the world independently: the structure of the lamp, for example, is independent of where it is placed on the table.
- Nested coordinates help manage complexity as well as promote reusability and simplify the transformations of objects composed of multiple primitive shapes

- Refer to the documentation: *https://threejs.org/docs/#api/en/core/Object3D*
- In Three.js 3D Objects have a ‘children’ array
- Use the method *.add(childObject)* to add a child to an object, i.e. to nest its transform
- Objects have a parent in the scene graph, and when you set their transforms (translation, rotation) , it’s relative to that parent’s local coordinate system

Three.js: Arbitrary Rotation, Cloning, Nested coordinates

- Threejs-5-partial-desk-lamp.html

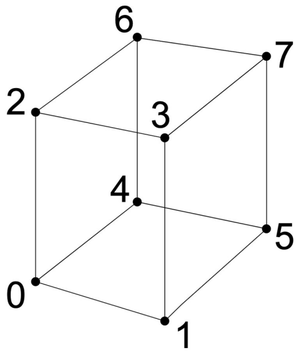
- This is a correctly set up hierarchy of nested objects, so we can:
  - Translate the base, and the 2 arms stay correct
  - Rotate the 1st arm, and the 2nd arm stays correct



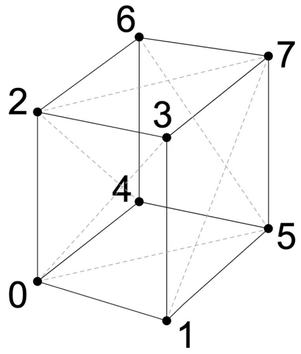
Three.js: Geometry beyond primitives

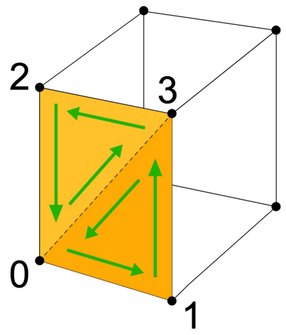
- Low Level Geometry" in Threejs:
  - Geometry objects consisting of vertices, faces, and normal (right-hand rule again)

- Some more useful techniques:
  - Extrusion Geometry (including Lathes)
  - Parametric Geometry
  - Loading geometry from file



8 vertices





12 triangular faces, each defined by 3 vertices, listed in anticlockwise order for front of face

Next Time

Ollscoil na Gaillimhe
University of Galway

- Shading with Texture Mapping
- Material Properties
- Animation and Interactivity

Ollscoil na Gaillimhe
University of Galway

Thank *you*